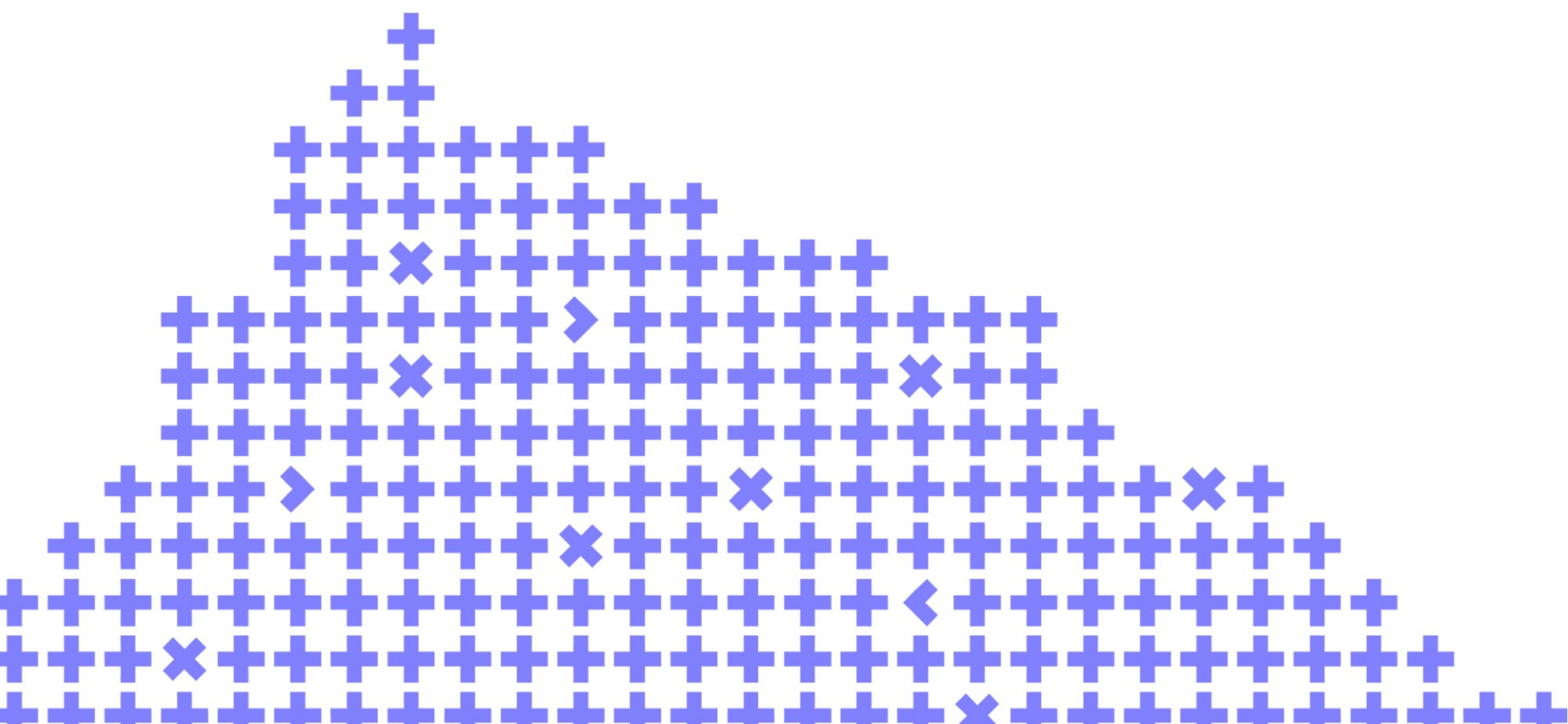


Transactional Queues in PostgreSQL

Igor Loban



Co-organizer

Yandex

Intro

- Body Level One
- 10 years in backend
- Body Level Two
- Body Level Three
- The last 6 years at 10koka.ai
- Body Level Four
- Lover of Python and PostgreSQL



About Toloka.ai

- Toloka is a data labelling service for ML
- 300 labels per second from real people
- 40 microservices
- Many clusters of PostgreSQL, MongoDB, ClickHouse, Redis
- SQS-like message brokers for async communication



About Toloka.ai

- Toloka is a data labelling service for ML
- 300 labels per second from real people
- 40 microservices
- Many clusters of PostgreSQL, MongoDB, ClickHouse, Redis
- SQS-like message brokers for async communication



Global

- **Body Level One**
 - Highlight the problem that forces to use transactional queues in PostgreSQL
- **Body Level Two**
 - Give a foundation to solve that problem in your projects
 - **Body Level Three**
 - **Body Level Four**
 - **Body Level Five**

Content

- 01** Why should we use transactional queues?
- 02** Transactional Outbox Design Pattern
- 03** PgQ and Eventuate
- 04** Recipe for Transactional Queues in PostgreSQL
- 05** Summary

01

Why should we use transactional queues?

Problem

- **Body Level One**
 - Distributed business transactions
- **Body Level Two**
 - also known as consistent changes in different services
- **Body Level Three**
- **Body Level Four**
- **Body Level Five**

Problem

- **Body Level One**
 - Distributed business transactions
- **Body Level Two**
 - also known as consistent changes in different services
- **Body Level Three**
 - Saga pattern is one of the solutions
- **Body Level Four**
 - The main challenge is **an atomic change in a local DB and sending a message to a broker**
- **Body Level Five**

Example 1

- `@Transactional`
`public void updateOrder(Order order) {`
 - `// transaction has been started`
 - `orderRepo.save(order);`
 - `writeToExternalQueue(order)`
 - `// commit can fail!`

`}`

Example 2

- `public void updateOrder(Order order) {`
 - Body Level One
 - `saveOrderInTx(order); // DB transaction inside the method`
 - Body Level Two
 - `writeToExternalQueue(order); // can fail!`
 - Body Level Three
 - Body Level Four
 - Body Level Five

Solution

- **Body Level One**
 - Background processor for retries
- **Body Level Two**
 - Boolean field “sent to queue” in an entity
 - **Body Level Three**
 - **Body Level Four**
 - **Body Level Five**

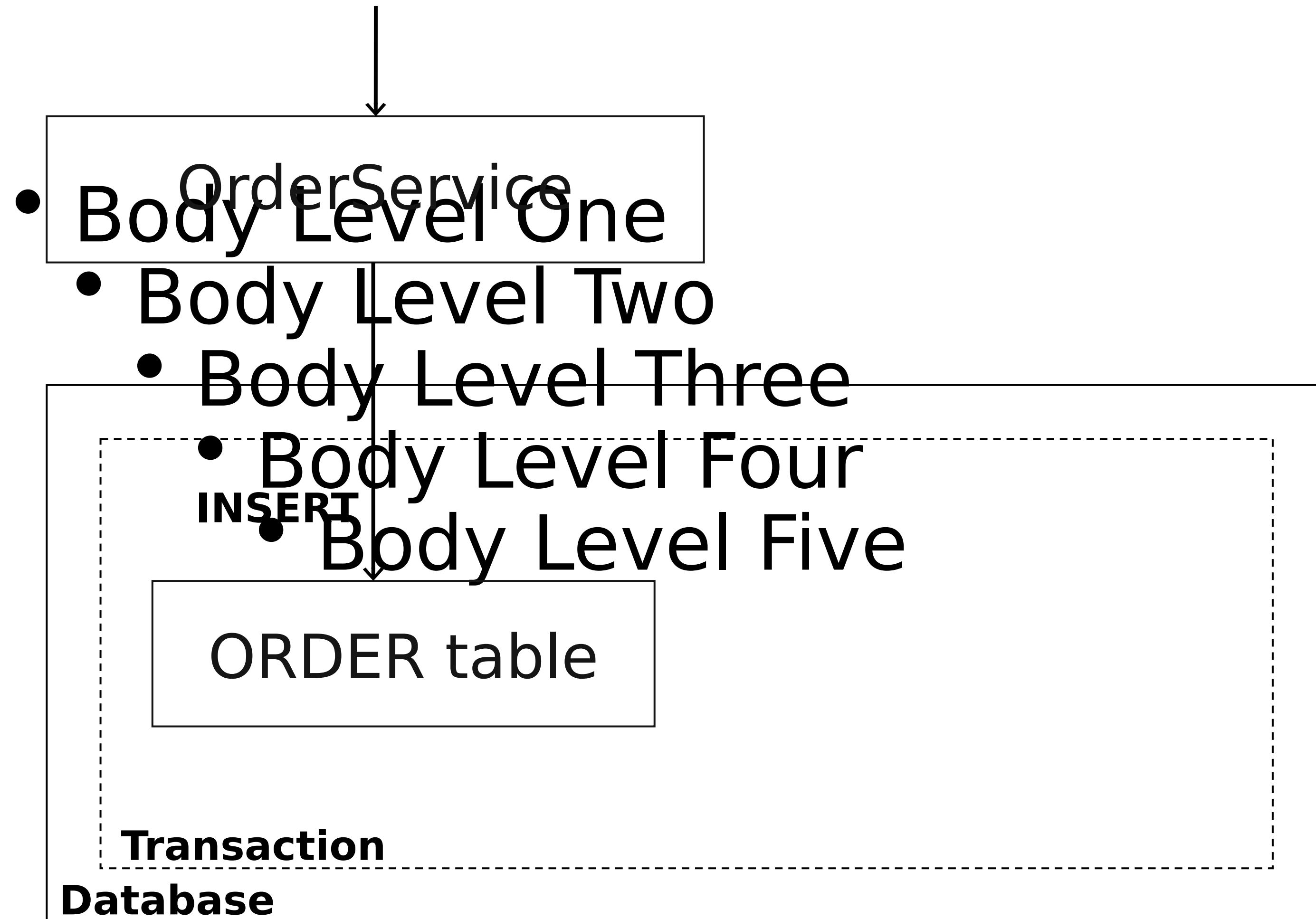
Solution

- **Body Level One**
 - Background processor for retries
- **Body Level Two**
 - Boolean field “sent to queue” in an entity
- **Body Level Three**
 - Problem 1: potential performance degradation
- **Body Level Four**
 - Problem 2: similar code in multiple places
- **Body Level Five**

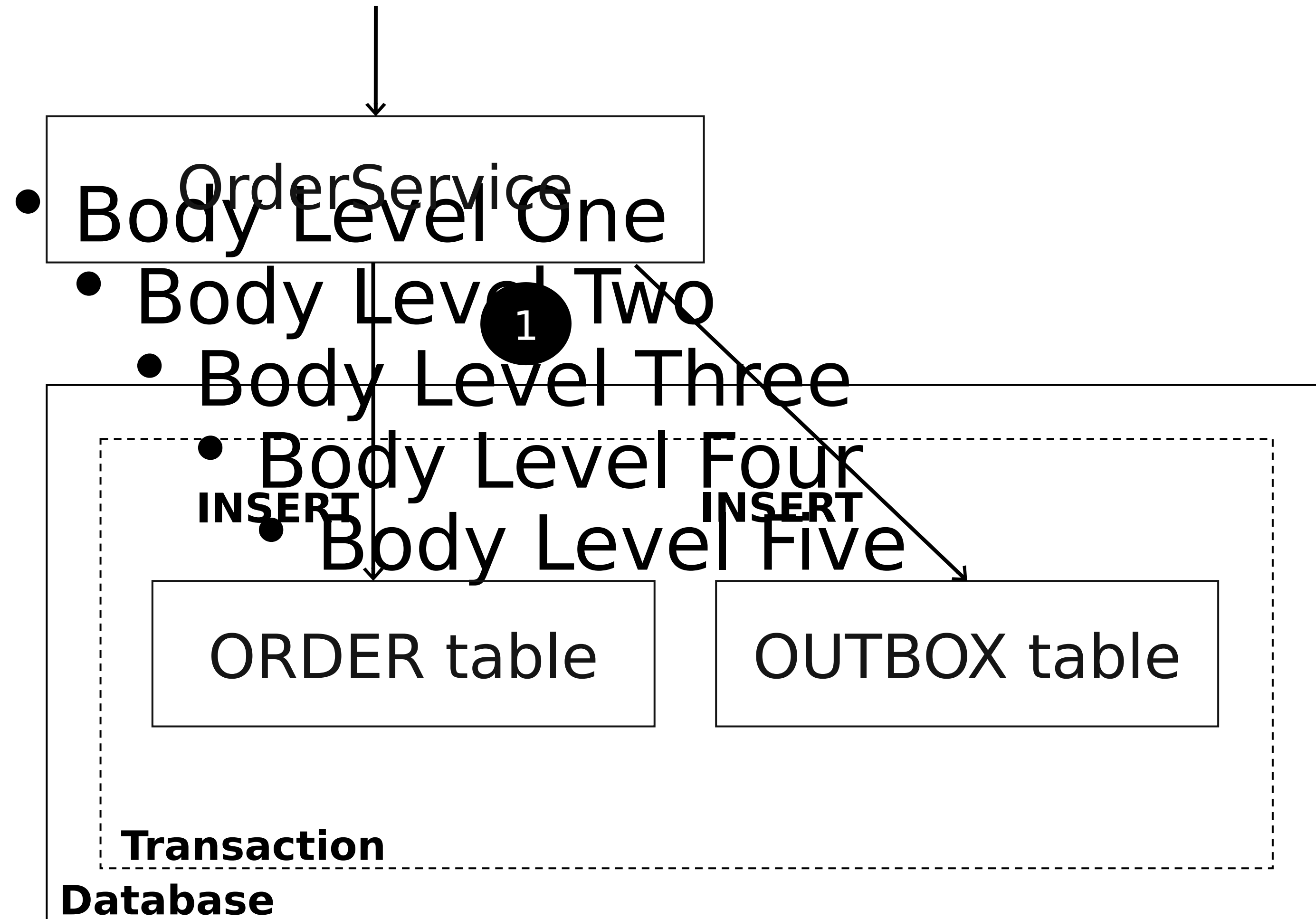
02

Transactional Outbox Design Pattern

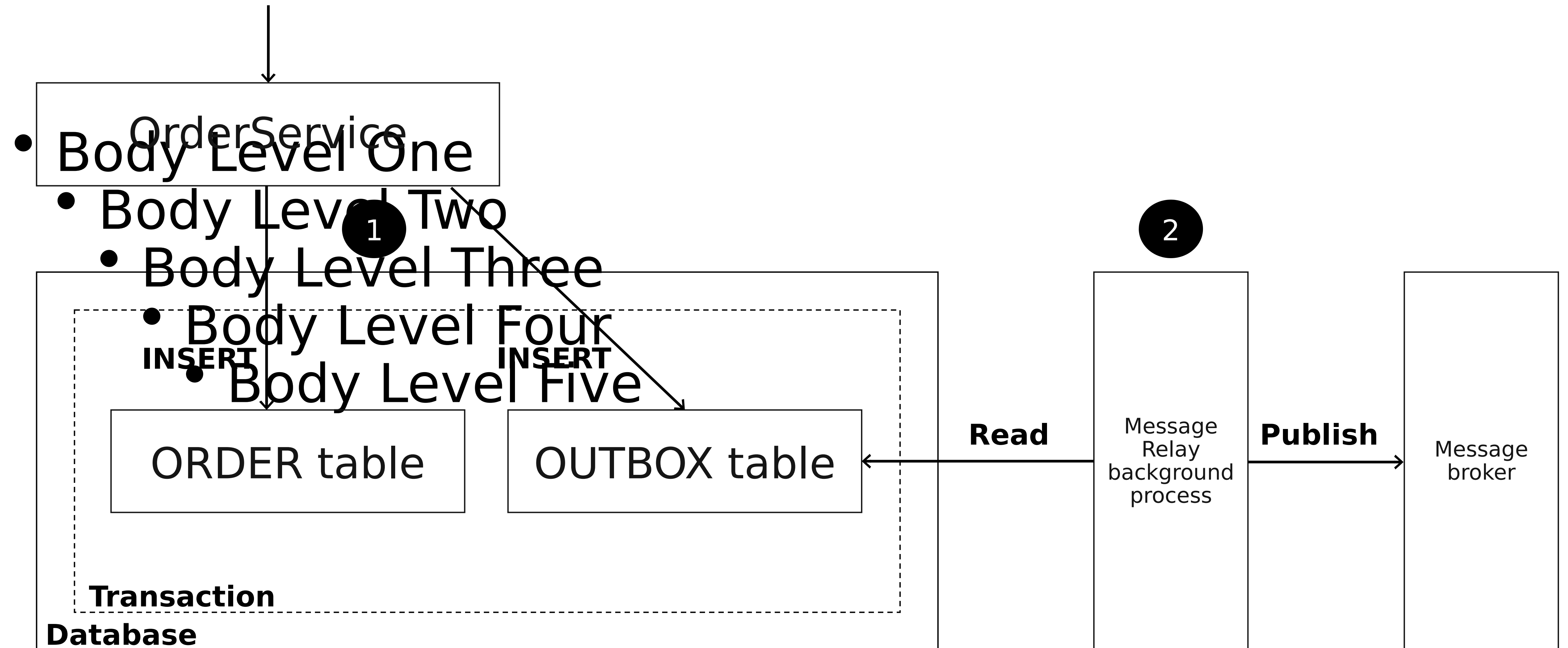
Transactional Outbox



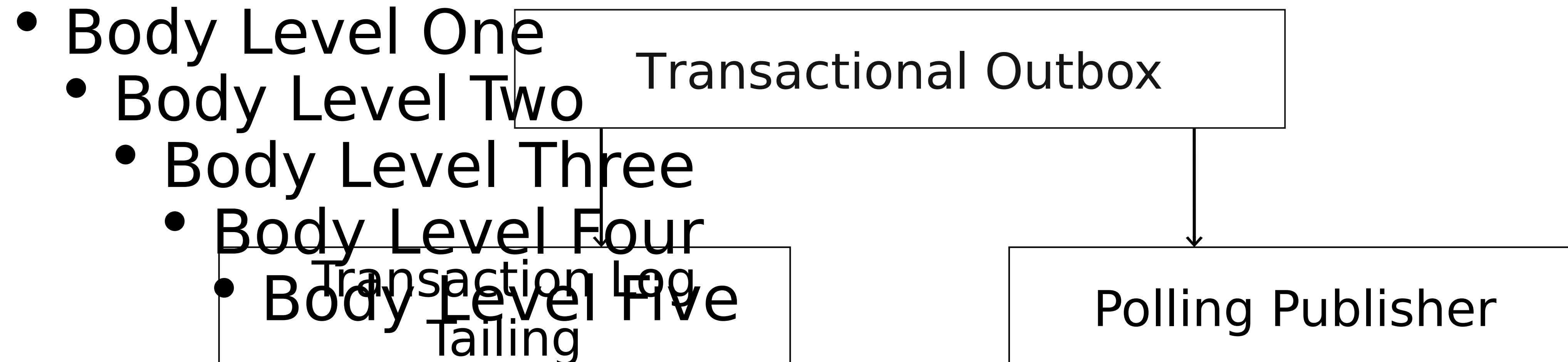
Transactional Outbox



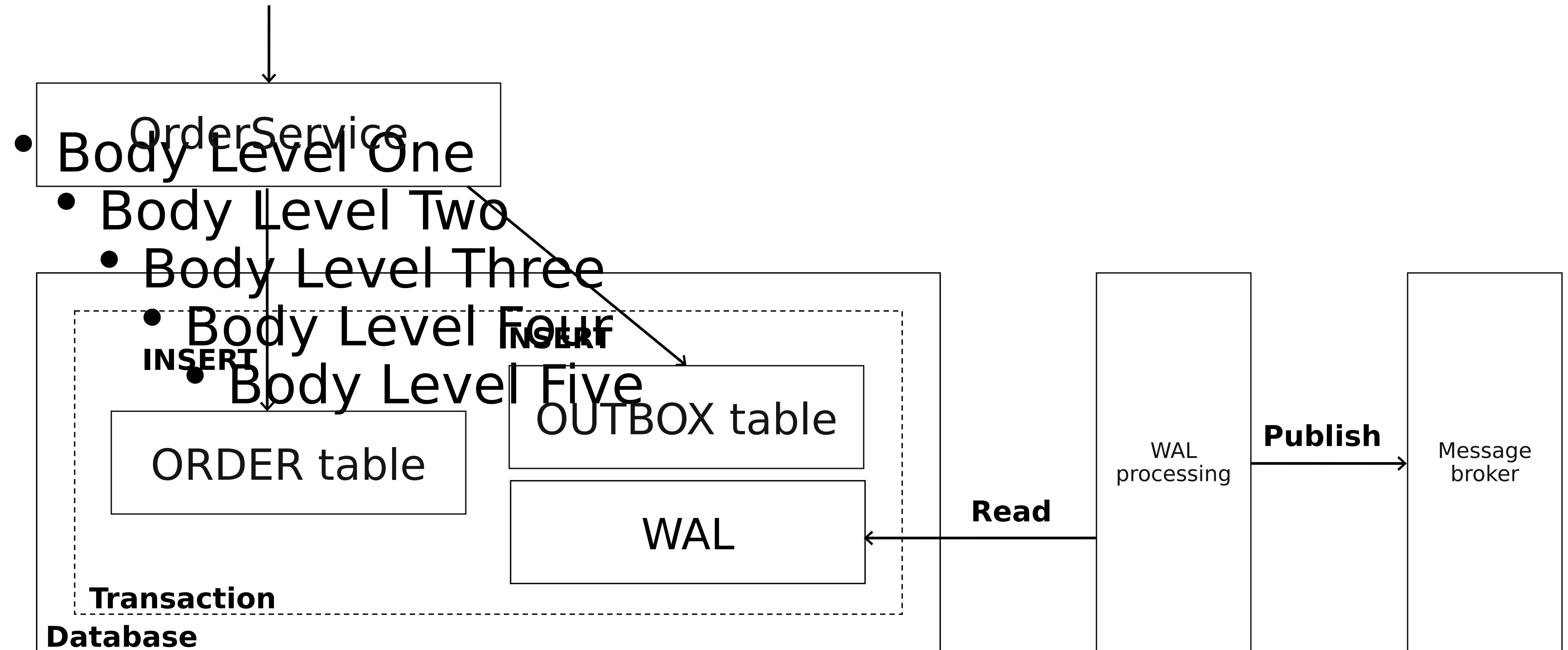
Transactional Outbox



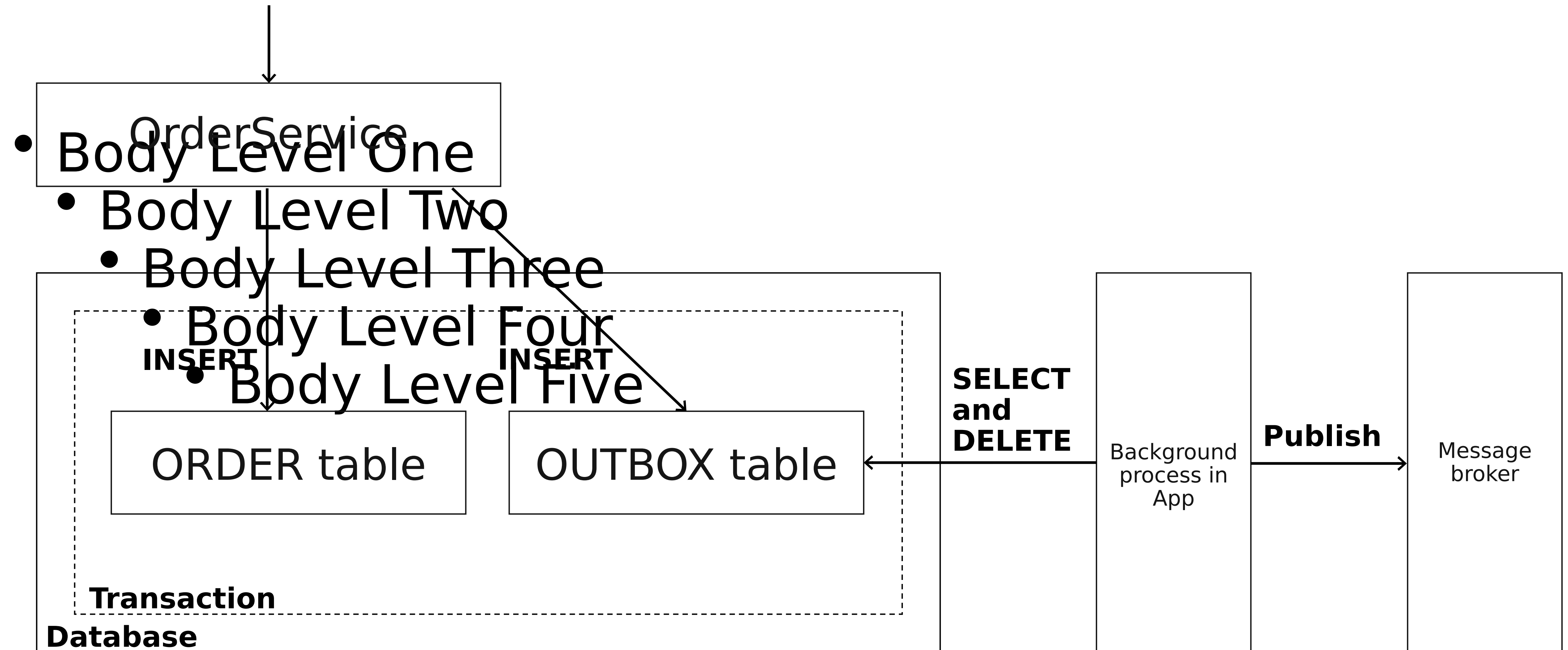
Transactional Outbox Subtypes



Transaction Log Tailing



Polling Publisher



Long DB Transactions

- Body Level One
 - Business logic of your application
- Body Level Two
 - Queries to an analytic replica (hot_standby_feedback)
- Body Level Three
 - Index building
- Body Level Four
 - pg_repack
- Body Level Five

Long DB Transactions

- Body Level One
 - Business logic of your application
- Body Level Two
 - Queries to an analytic replica (hot_standby_feedback)
- Body Level Three
 - Index building
- Body Level Four
 - pg_repack
- Body Level Five


Long DB Transactions


- Body Level One
 - Business logic of your application
- Body Level Two
 - Queries to an analytic replica (hot_standby_feedback)
- Body Level Three
 - Index building
- Body Level Four
 - pg_repack
- Body Level Five


03


PgQ and Eventuate

PgQ

 [pgq](#) / [pgq](#) Public


 Watch





 **Starred** **166**



[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Security](#) [Insights](#)


[master](#) [Go to file](#) [Add file](#) [Code](#) [About](#)


 **markokr** Update versions in RE... ... ✓ on Aug 9 🕒 67


	.github/wo...	ci: update matrix	4 months ago
	docs	v3.5	4 months ago
	expected	jsontriga: switch backup ro...	4 months ago
	functions	Show extension version in ...	2 years ago


Generic Queue for PostgreSQL

[queue](#) [postgresql](#)

 Readme


 ISC license


 **166** stars


 **11** watching


- Extension for PostgreSQL
- Full-fledged message broker based on PostgreSQL
- Requires process-daemon pgqd

PgQ

 [pgq](#) / [pgq](#) Public


 Watch





 **Starred** **166**



[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Security](#) [Insights](#)


[master](#) [Go to file](#) [Add file](#) [Code](#) [About](#)


 **markokr** Update versions in RE... ... ✓ on Aug 9 🕒 67


	.github/wo...	ci: update matrix	4 months ago
	docs	v3.5	4 months ago
	expected	jsontriga: switch backup ro...	4 months ago
	functions	Show extension version in ...	2 years ago


Generic Queue for PostgreSQL

[queue](#) [postgresql](#)

 Readme

 ISC license

 **166** stars

 **11** watching

- Extension for PostgreSQL
- Full-fledged message broker based on PostgreSQL
- Requires process-daemon pgqd

Eventuate

eventuate-tram / eventuate-tram-core Public

Starred 841

Code Issues 84 Pull requests 1 Act

master

Go to file Add file Code

About

Transactional messaging for microservices

Readme View license 841 stars 61 watching 165 forks

cer Fixed eventuate-foundation/eventuate-common#125	✓ on Oct 19	🕒 587
.circleci	Upgraded Orb to address script execution iss...	4 months ago
activemq	Fixed activemq user/password	4 years ago
buildSrc	Upgraded Gradle Wrapper, eliminated redund...	2 years ago
eventuate-tram-aggr...	Upgraded Gradle Wrapper, eliminated redund...	2 years ago

- Framework for Transactional Outbox and Saga patterns
- PostgreSQL and MySQL – Transaction Log Tailing
- SQL Server – Polling Publisher

04

Recipe for Transactional Queues in PostgreSQL

Limitations

- Body Level One
 - At-least-once delivery
- Body Level Two
 - No ordering guarantee
- Body Level Three
 - Body Level Four
 - Body Level Five

Stress-test

- **Body Level One**
 - Disclaimer: it is not a benchmark
- **Body Level Two**
 - 70 writers (1000 events per second)
- **Body Level Three**
 - 5 readers (2000 events per second)
- **Body Level Four**
 - Sync replica with 50 ms delay (recovery_min_apply_delay)
- **Body Level Five**
 - Begin a long transaction and wait for 5 min

Stress-test

- **Body Level One**
 - Disclaimer: it is not a benchmark
- **Body Level Two**
 - 70 writers (1000 events per second)
- **Body Level Three**
 - 5 readers (2000 events per second)
- **Body Level Four**
 - Sync replica with 50 ms delay (recovery_min_apply_delay)
- **Body Level Five**
 - Begin a long transaction and wait for 5 min

Stress-test

- **Body Level One**
 - Disclaimer: it is not a benchmark
- **Body Level Two**
 - 70 writers (1000 events per second)
- **Body Level Three**
 - 5 readers (2000 events per second)
- **Body Level Four**
 - Sync replica with 50 ms delay (recovery_min_apply_delay)
- **Body Level Five**
 - Begin a long transaction and wait for 5 min

TitFor UPDATE

The OUTBOX table

- Body Level One
 - Body Level Two
 - Body Level Three
 - Body Level Four
 - Body Level Five

```
CREATE TABLE queue_buffer
```

```
id BIGSERIAL PRIMARY KEY
```

```
payload TEXT
```

```
);
```

Tip FOR UPDATE

Readers lock rows, sleep, and delete them in a single transaction

- Body Level One
- Body Level Two
- Body Level Three
- Body Level Four
- Body Level Five

BEGIN

SELECT id, payload FROM queue_buffer

ORDER BY id

FOR UPDATE

LIMIT ?;

...

DELETE FROM queue_buffer WHERE id IN (...);

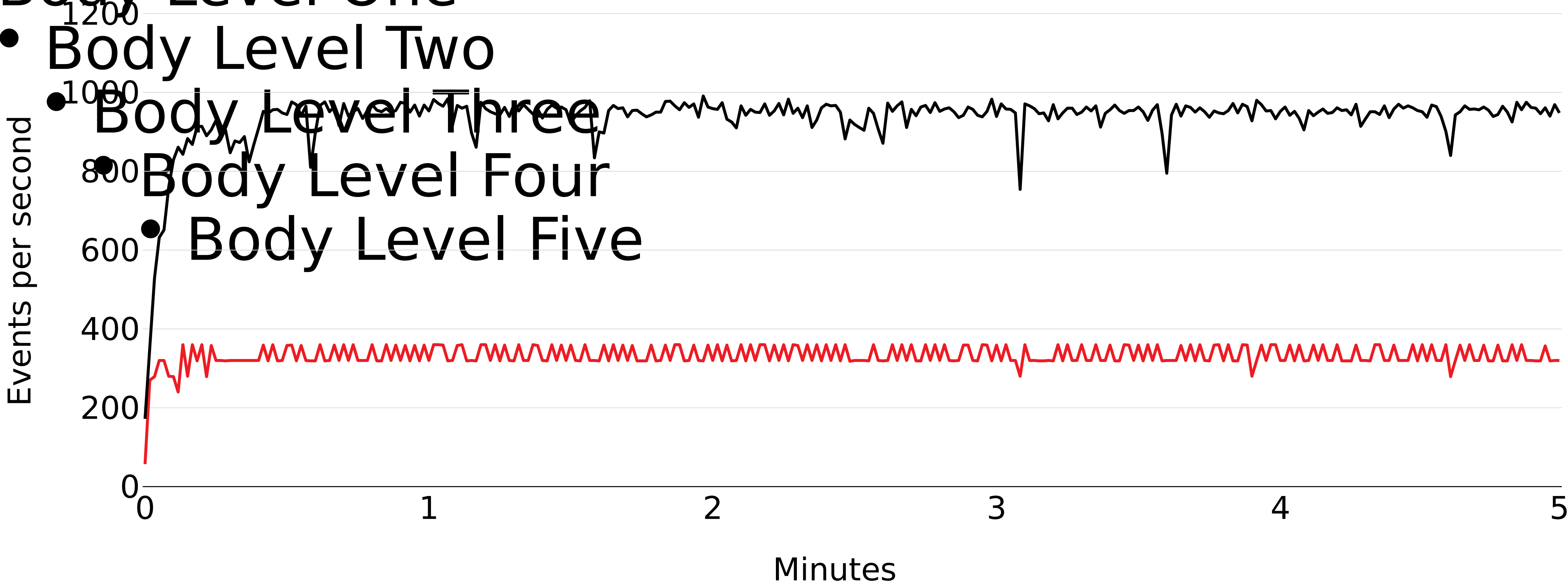
COMMIT;

Tithe FOR UPDATE

Throughput WITHOUT a long transaction

— Read — Write

- Body Level One
- Body Level Two
- Body Level Three
- Body Level Four
- Body Level Five

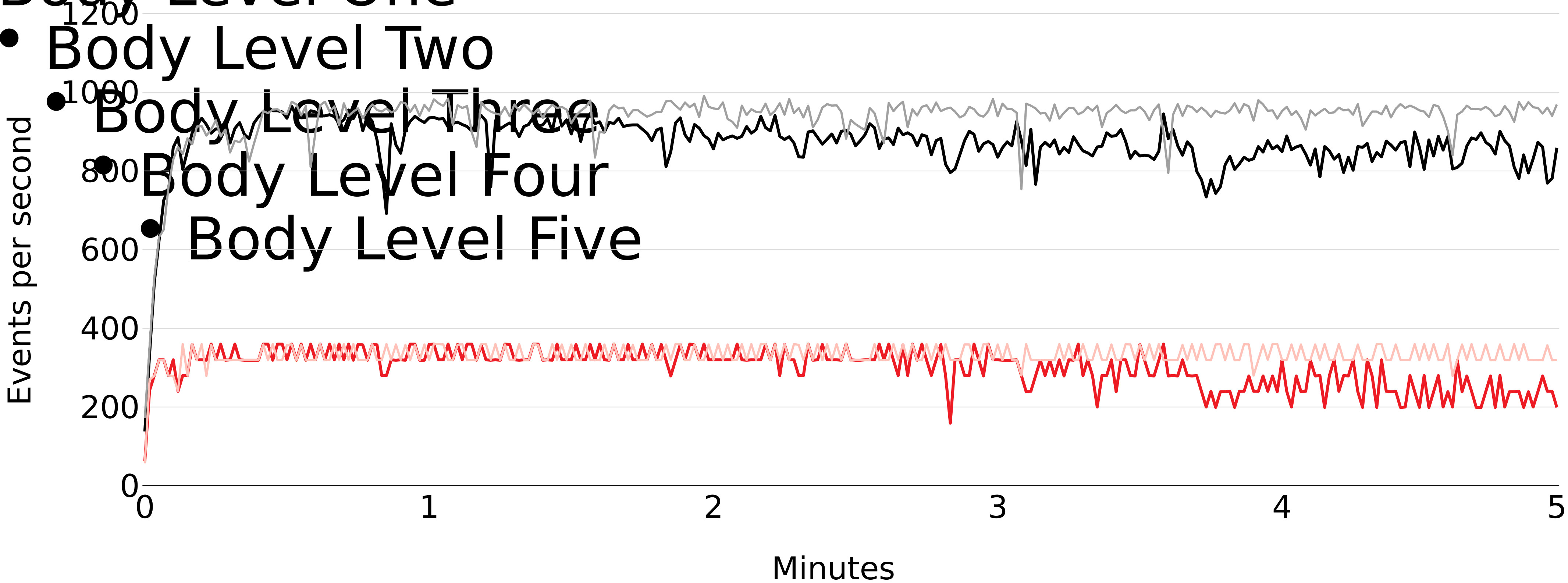


Tithe FOR UPDATE

Throughput WITH a long transactions

— Read — Write — Old Read — Old Write

- Body Level One
- Body Level Two
- Body Level Three
- Body Level Four
- Body Level Five



2its SKIP LOCKED

- Body Level One
 - SELECT ... FOR UPDATE SKIP LOCKED
- Body Level Two
 - The reader concurrency problem is solved
- Body Level Three
 - No order guarantee
- Body Level Four
 - Body Level Five

Zits SKIP LOCKED

Throughput WITHOUT a long transaction

— Read — Write

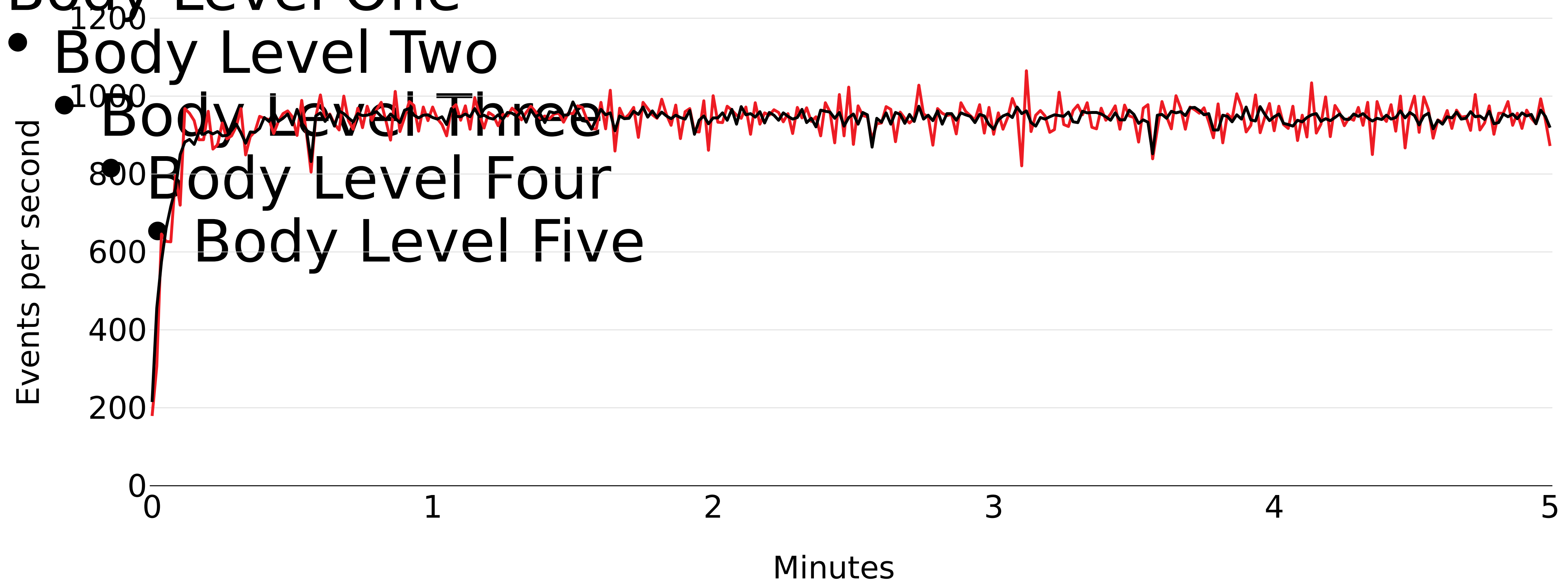
- Body Level One

- Body Level Two

- Body Level Three

- Body Level Four

- Body Level Five



Zits SKIP LOCKED

Throughput WITH a long transaction

— Read — Write

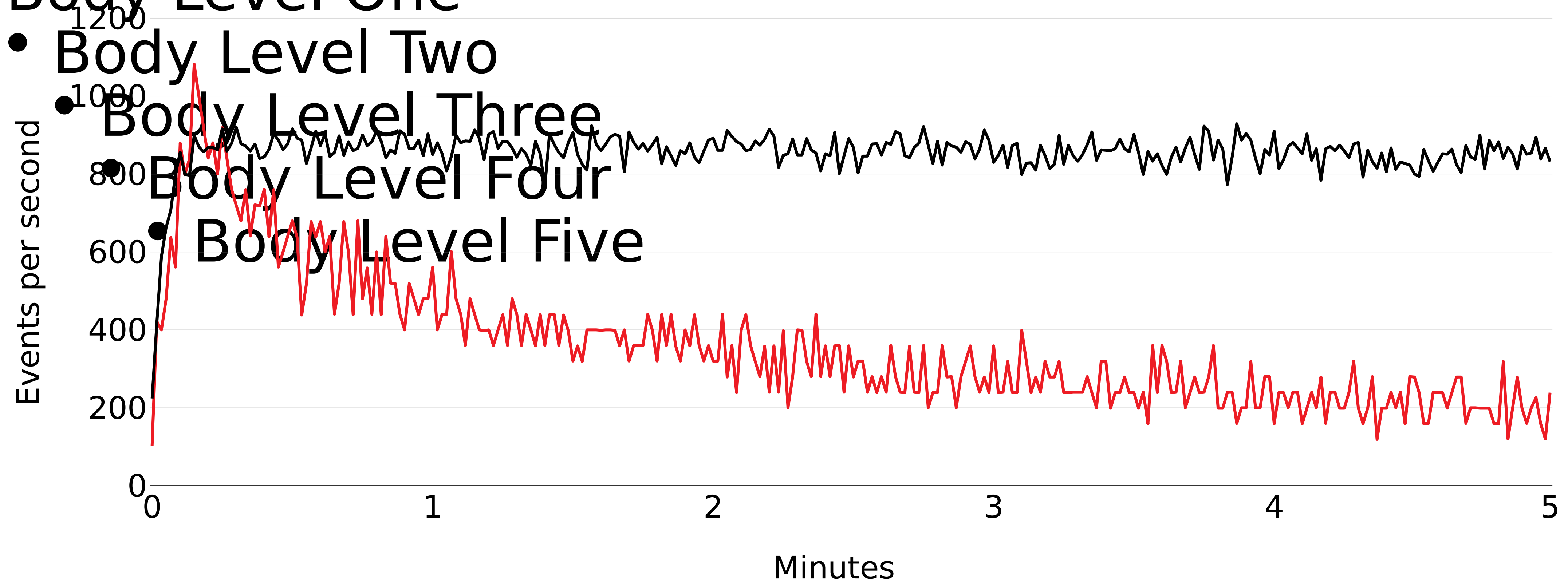
- Body Level One

- Body Level Two

- Body Level Three

- Body Level Four

- Body Level Five



3. Bitwise WHERE id > :prev_id

- Body Level One
 - SELECT ... WHERE id > :prev_id ... FOR UPDATE SKIP LOCKED
- Body Level Two
 - prev_id is the last processed ID
- Body Level Three
 - Every 10th query will reset prev_id
- Body Level Four
 - Body Level Five

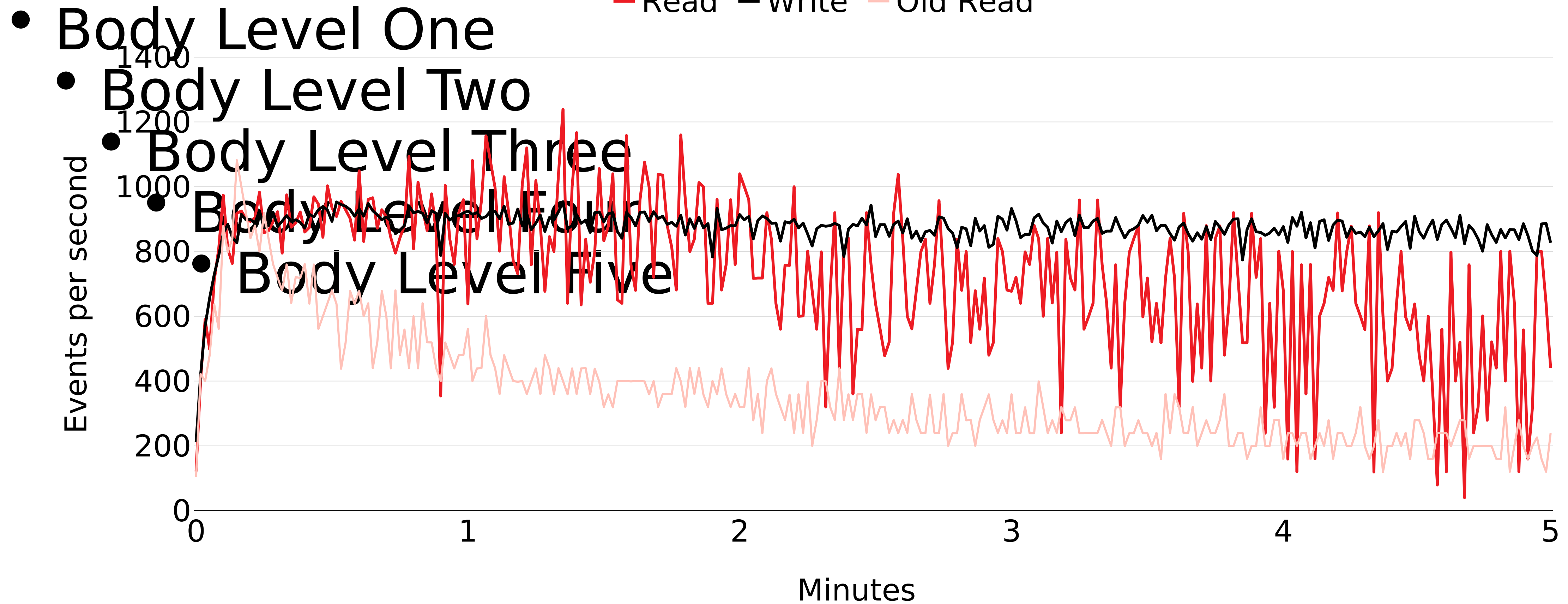
3. Bitwise WHERE id > :prev_id

- Body Level One
 - SELECT ... WHERE id > :prev_id ... FOR UPDATE SKIP LOCKED
- Body Level Two
 - prev_id is the last processed ID
- Body Level Three
 - Every 10th query will reset prev_id
- Body Level Four
 - Body Level Five

3 WHERE id > :prev_id

Throughput WITH a long transaction

— Read — Write — Old Read



4.1 SET LOCAL synchronous_commit TO OFF

Throughput WITH a long transaction

— Read — Write — Old Read

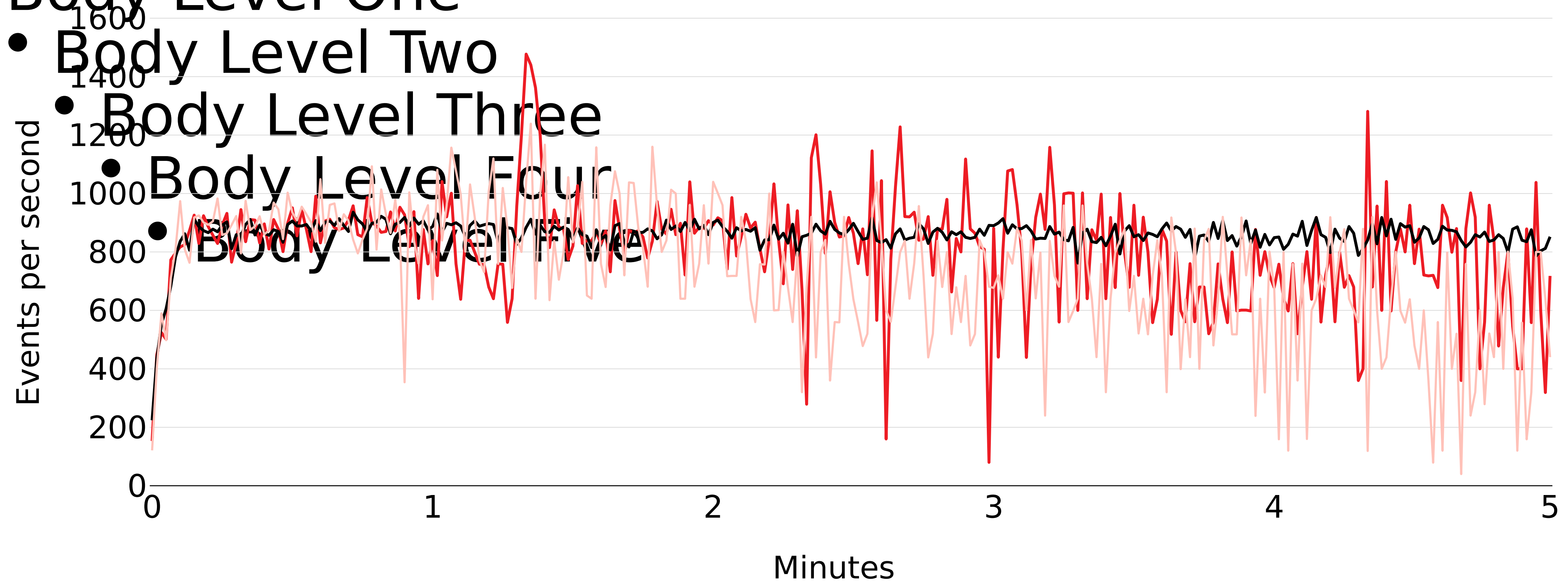
- Body Level One

- Body Level Two

- Body Level Three

- Body Level Four

- Body Level Five



4.1 SET LOCAL synchronous_commit TO OFF

Throughput WITH a long transaction

— Read — Write — Old Read

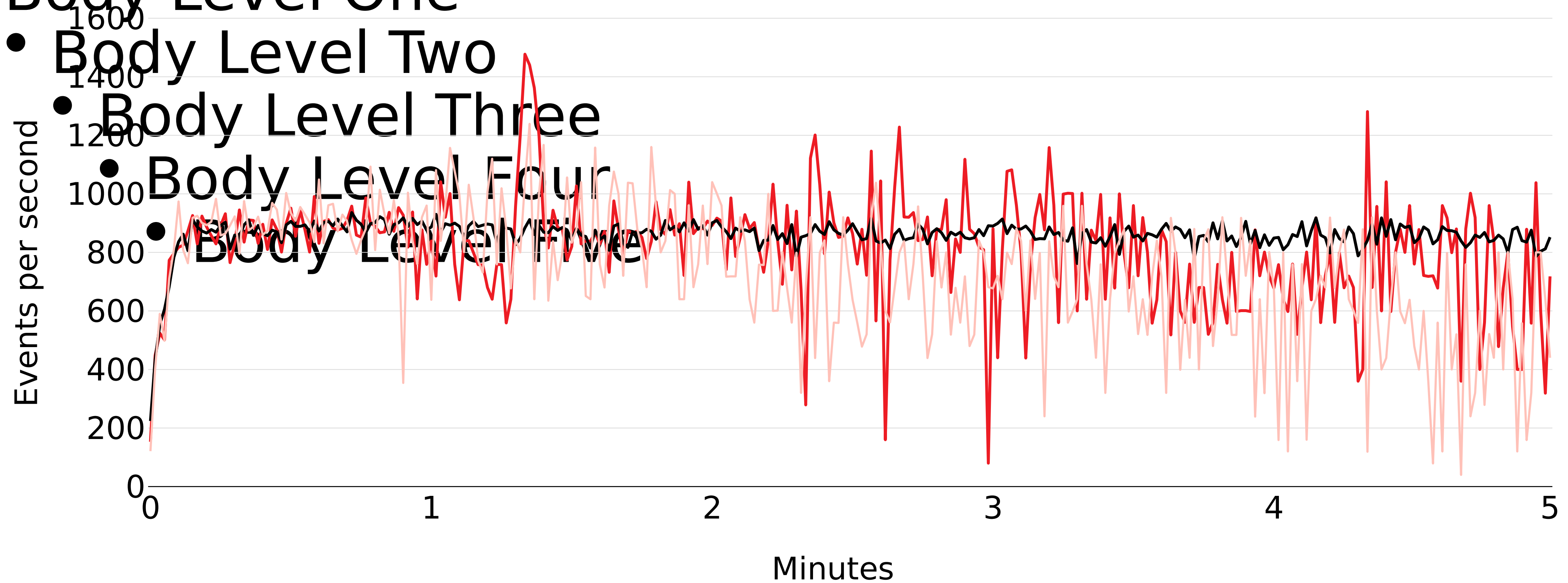
- Body Level One

- Body Level Two

- Body Level Three

- Body Level Four

- Body Level Five



Sitter RUNCATE

- **Body Level One**
 - TRUNCATE is not MVCC safe
- **Body Level Two**
 - Three queue buffer tables instead of one
- **Body Level Three**
 - Run TRUNCATE every 10 seconds
- **Body Level Four**
 - **Body Level Five**

Sitter RUNCATE

- **Body Level One**
 - TRUNCATE is not MVCC safe
- **Body Level Two**
 - Three queue buffer tables instead of one
- **Body Level Three**
 - Run TRUNCATE every 10 seconds
- **Body Level Four**
 - **Body Level Five**

SitterUNCATE

Writers will insert a message, if they can take a shared lock. $N = 1, 2, 3$

- Body Level One

- Body Level Two

- Body Level Three

- Body Level Four

- Body Level Five

COMMIT;

Sitter RUNCATE

The TRUNCATE background process. N = 1, 2, 3

- Body Level One

BEGIN;

- Body Level Two

SET LOCAL lock_timeout = '30s'; SET LOCAL statement_timeout = '30s';

- Body Level Three

SELECT pg_advisory_xact_lock(N);

- Body Level Four

LOCK TABLE queue_buffer_N IN ACCESS EXCLUSIVE MODE;

- Body Level Five

CREATE TEMPORARY TABLE queue_buffer_N_copy ON COMMIT DROP AS SELECT * FROM queue_buffer_N
WITH DATA;

TRUNCATE queue_buffer_N;

INSERT INTO queue_buffer_N SELECT * FROM queue_buffer_N_copy;

COMMIT;

SitterUNCATE

Throughput WITHOUT a long transaction

— Read — Write — Old Write

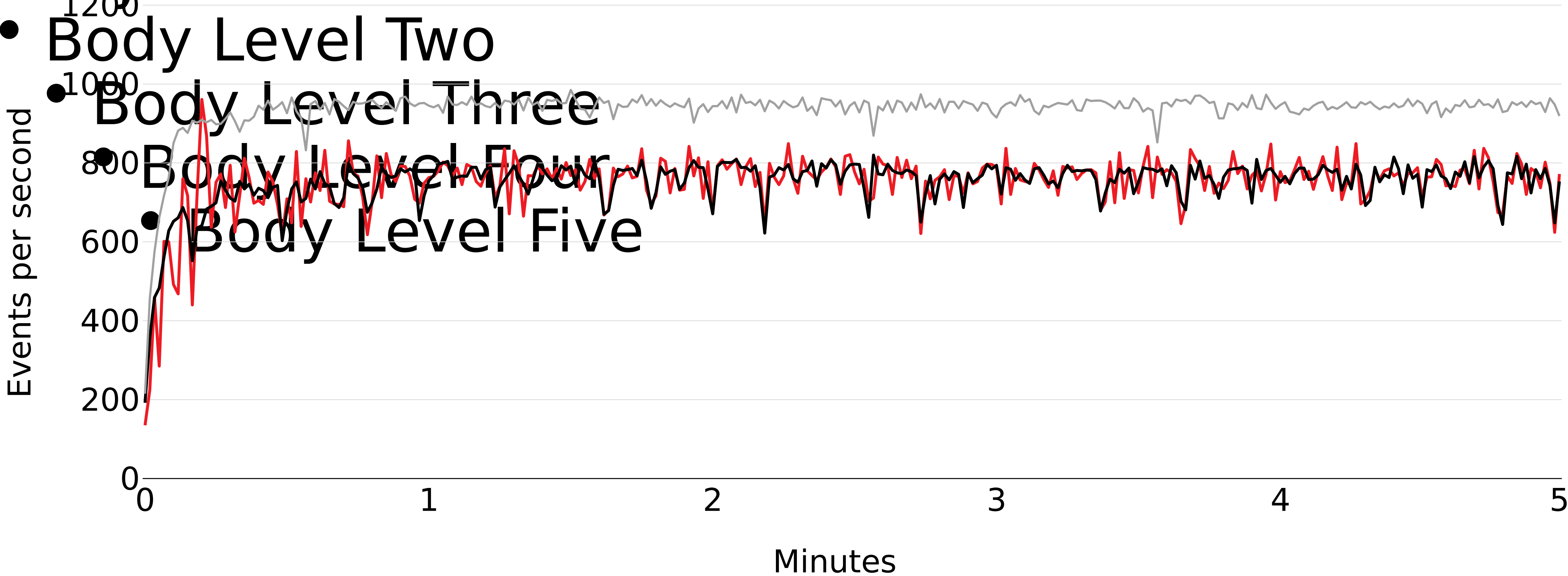
- Body Level One

- Body Level Two

- Body Level Three

- Body Level Four

- Body Level Five



SitterUNCATE

Throughput WITH a long transaction (but without TRUNCATE)

— Read — Write

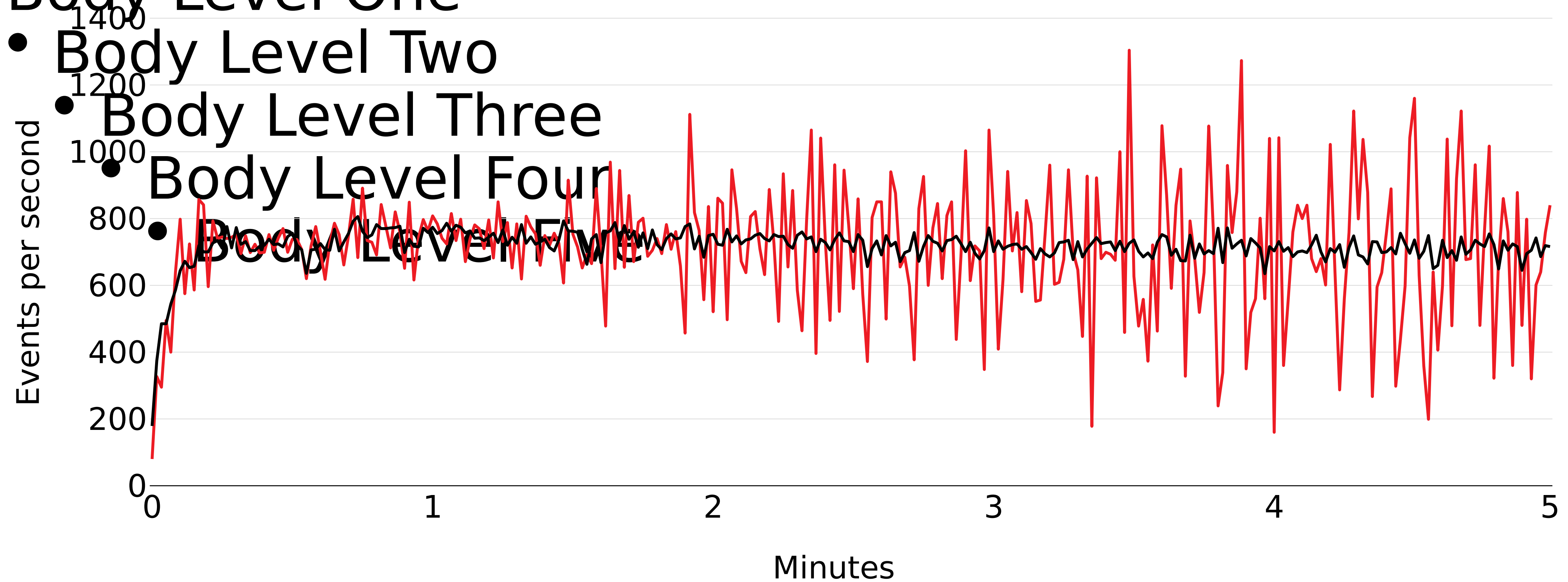
- Body Level One

- Body Level Two

- Body Level Three

- Body Level Four

- Body Level Five

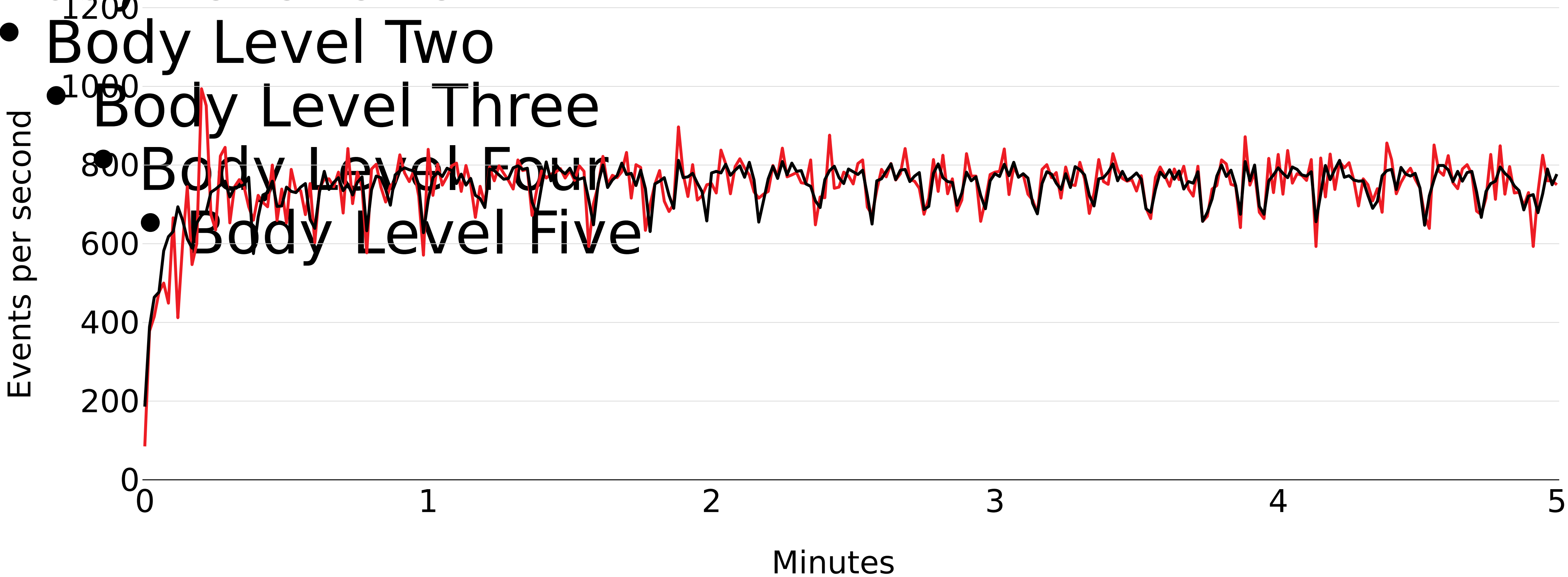


SitterUNCATE

Throughput WITH a long transaction

— Read — Write

- Body Level One
- Body Level Two
- Body Level Three
- Body Level Four
- Body Level Five



05

Summary

Playground

Toloka / pg-queue-playground

watch 4

Fork 0

Starred 2

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

main

1 branch

0 tags

Go to file

Add file

<> Code

Igor Loban Initial commit

2ecc4a8 2 hours ago

1 commit

gradle/wrapper	Initial commit	2 hours ago
pg-configs	Initial commit	2 hours ago
src	Initial commit	2 hours ago
.gitattributes	Initial commit	2 hours ago
.gitignore	Initial commit	2 hours ago

About

Playground for transactional queues in PostgreSQL

Readme

View license

2 stars

4 watching

0 forks



Summary

- Body Level One
 - We found out when transactional queues in DBs are needed
- Body Level Two
 - Transactional Outbox pattern
- Body Level Three
 - PgQ and Eventuate
- Body Level Four
 - Performance tricks
- Body Level Five
 - TRUNCATE against long transactions



Summary

- Body Level One
 - We found out when transactional queues in DBs are needed
- Body Level Two
 - Transactional Outbox pattern
- Body Level Three
 - PgQ and Eventuate
- Body Level Four
 - Performance tricks
- Body Level Five
 - TRUNCATE against long transactions



Summary

- Body Level One
 - We found out when transactional queues in DBs are needed
- Body Level Two
 - Transactional Outbox pattern
- Body Level Three
 - PgQ and Eventuate
- Body Level Four
 - Performance tricks
- Body Level Five
 - TRUNCATE against long transactions



Kindly remind about **metrics**, **alerts**, and **load testing** for your solutions

Links

Design patterns

• Body Level One

- <https://microservices.io/patterns/data/transactional-outbox.html>

• Body Level Two

- <https://microservices.io/patterns/data/transaction-log-tailing.html>

• Body Level Three

- <https://microservices.io/patterns/data/polling-publisher.html>

- <https://microservices.io/patterns/data/saga.html>

• Body Level Four

• Body Level Five

PgQ and Eventuate

- https://www.pgcon.org/2009/schedule/attachments/91_pgq.pdf

- <https://habr.com/ru/post/483014/>

- <https://github.com/eventuate-tram>

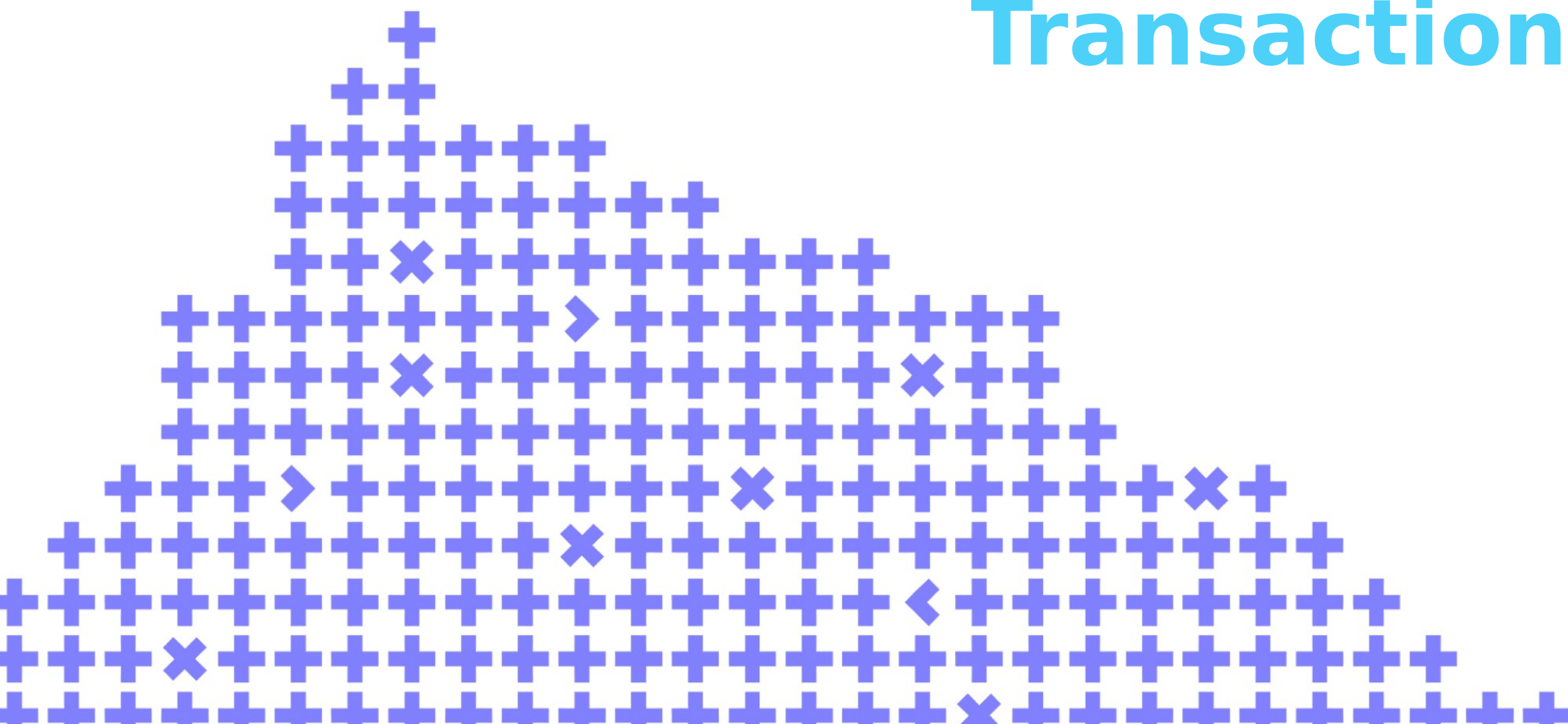
Leave your feedback!

You can rate the talk and
give feedback on what
you've liked or what
could be improved



Transactional Queues in PostgreSQL

Igor Loban



Co-organizer

Yandex